Jet Propulsion Laboratory
California Institute of Technology

Applying Model Based Design to Multicore Flight Software

Amalaye Oyake

Jet Propulsion Laboratory
California Institute of Technology

INTRODUCTION

- **Writing applications that take advantage of multicore processors requires an understanding of the multiple dimensions of the problem domain of the application and processor characteristics, the features provided by the operating system and the correct design patterns that can make best use of the multiprocessing capabilities.**

- **This presentation looks at some concepts to migrate a flight software application to a multicore (in this case VxWorks SMP) environment.**

GOAL

- A driving requirement for JPL will be to preserve its install base of useful legacy (VxWorks based) software and migrate these applications to a multicore environment.
- There is a knowledge base of existing technology that is often 'reused' between missions – attitude control, navigation, proximity and deep space communication etc…
- Further more the increased performance capabilities will allow JPL to meet the needs of future mission requirements. Some of these future mission needs - interferometry, (space and earth based) radar processing, formation flying, video from space, autonomy, fault protection and spacecraft communication (middleware), onboard databases, autonomous responses to processed sensor data and the like.

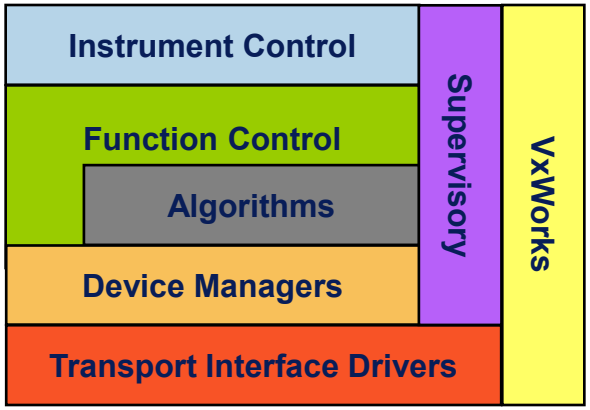
RTC FSW to VxWorks SMP

- For this work we are proposing, the JPL Real Time Control FSW framework(RTC FSW) …
- RTC is designed as a modular flight software architecture for controlling spacecraft with high performance – formation flying, interferometers, robotic systems.
- The RTC FSW application was written in C++ and runs on Linux and VxWorks.
- A significant amount of code is automatically generated from UML statecharts.
- It is designed to support a multiprocessor environment – real time computers connected via a messaging mechanism … we want to move to a multicore environment.
- It supports the notion of a software component.

Modular Flight Software Description

SIM RTC Architecture

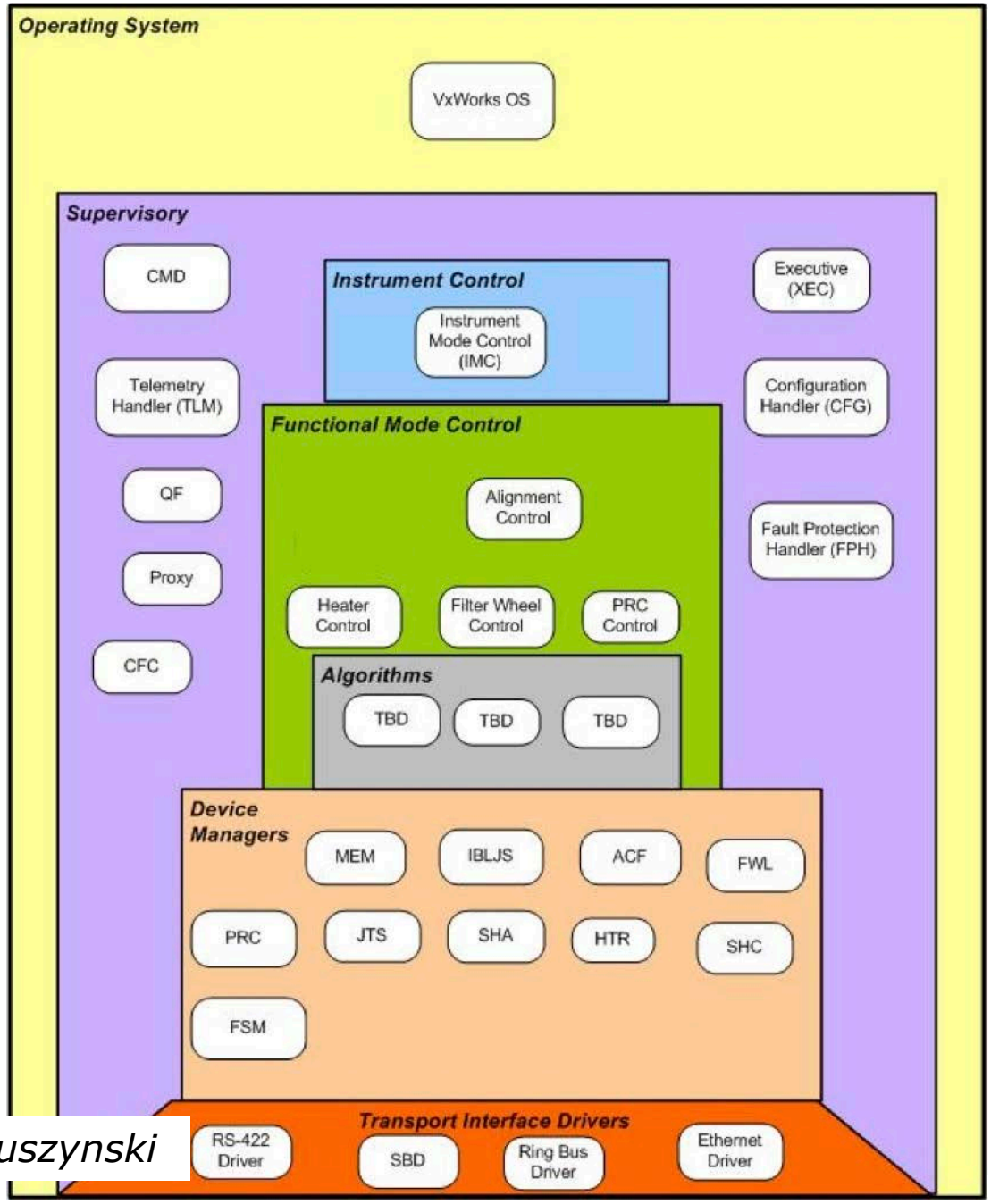
- **Same layered core components**
- **Same communications interfaces**
- **Same synchronized RTI tasking (but faster)**
- **State based development with autocoder**
- **Same dynamic configuration system**
- **Communication**



** Source: Marek Tuszynski*

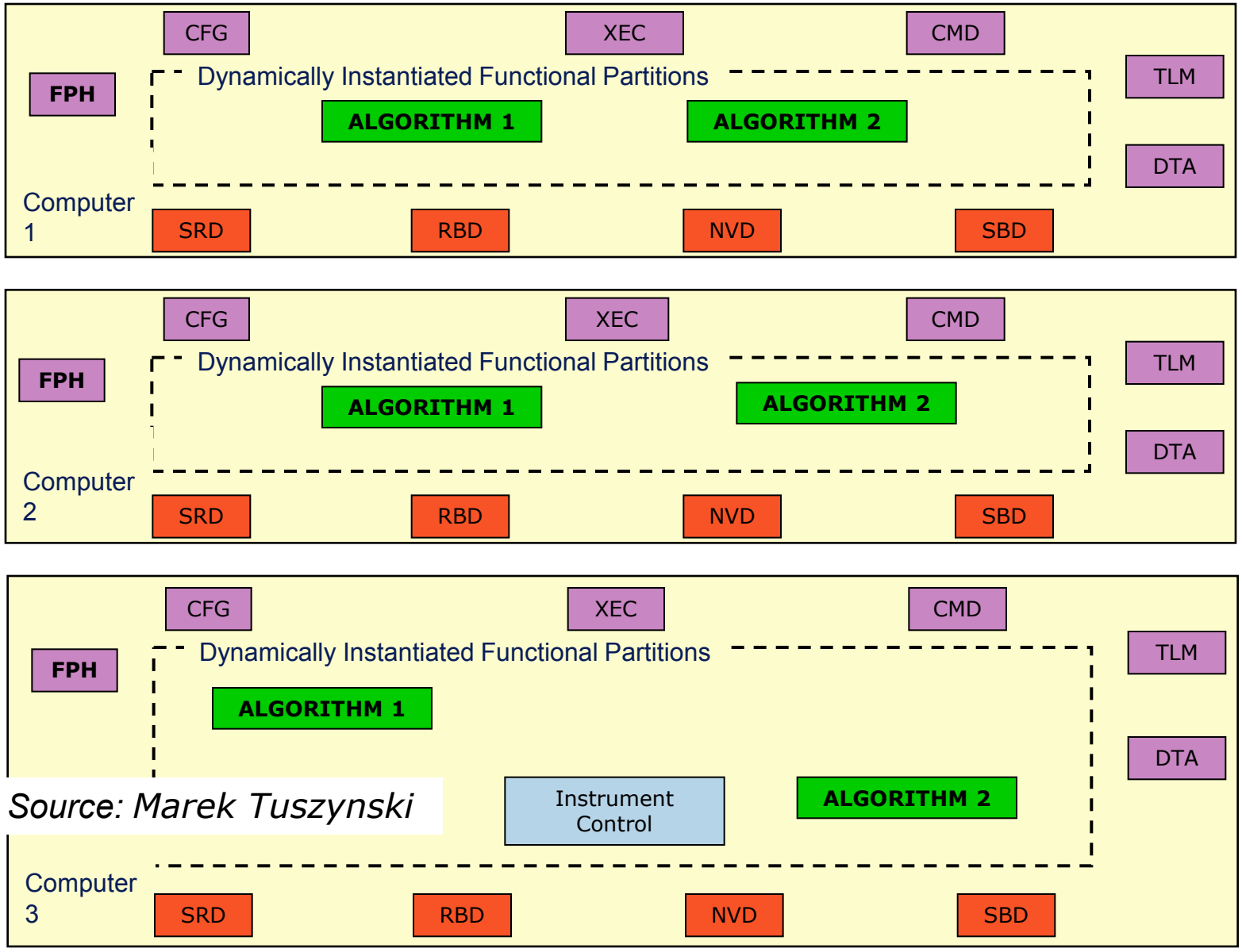
Architecture - CCDH SW Layers and Components

- **Communication limited to connected layers**
- **Encapsulation and data hiding isolate the application from a specific operating system and device hardware protocol**
- **Units in layers know only about layer directly under them**
  - **Don't know or care who calls them**
  - **Only care one layer down**
  - **May publish status events**



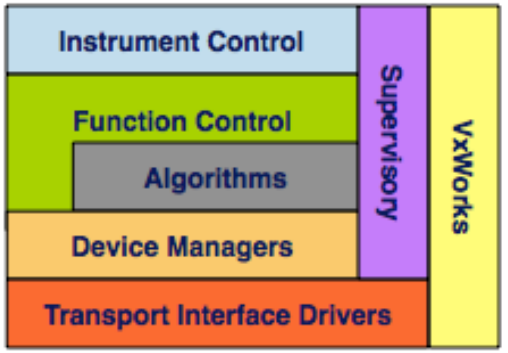
** Source: Marek Tuszynski*

Example FSW Distributed System

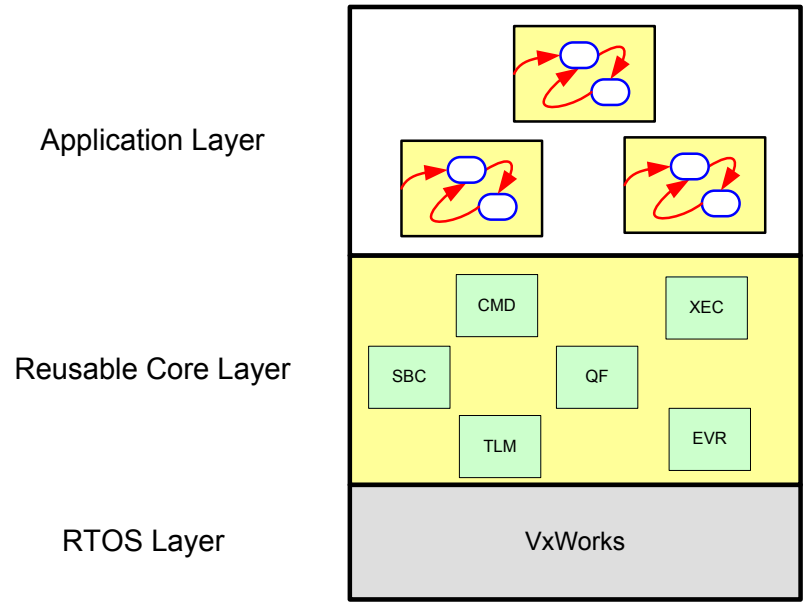


SIM RTC: Core SW: Architecture

- **The Core Layer provides a level of abstraction higher than the real-time Operating System.**
- **The Core Layer provides common real-time software capabilities such as schedule control, commanding, telemetry and event reporting.**
- **The Core Layer provides a state-based framework for the development of an Application specified as a collection of concurrently running state-machines.**



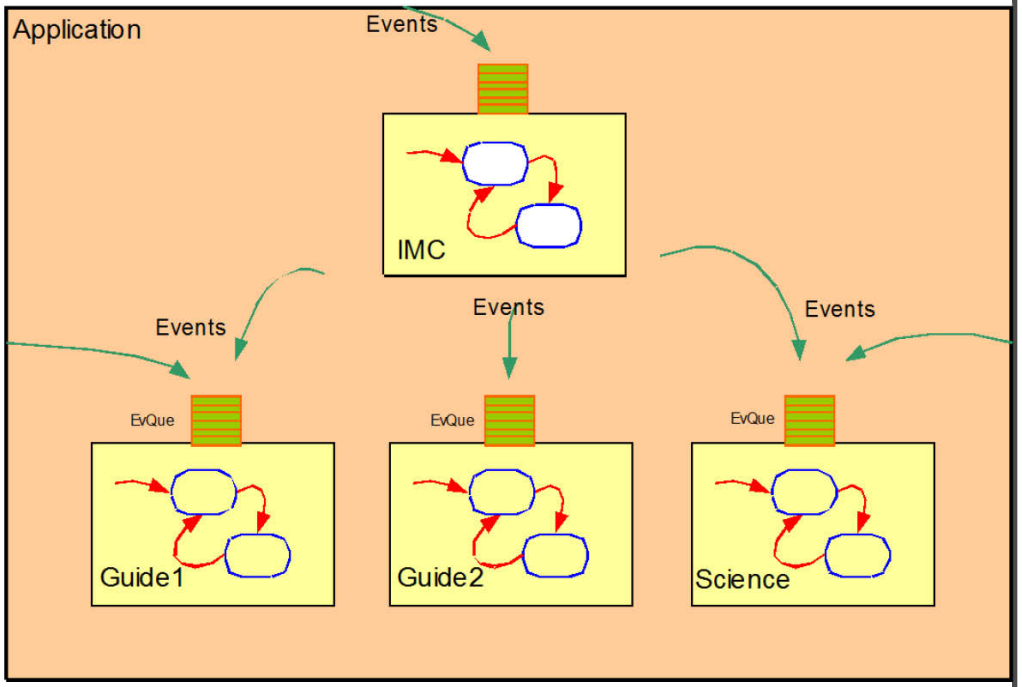
Layered architecture simplifies modules' interfaces



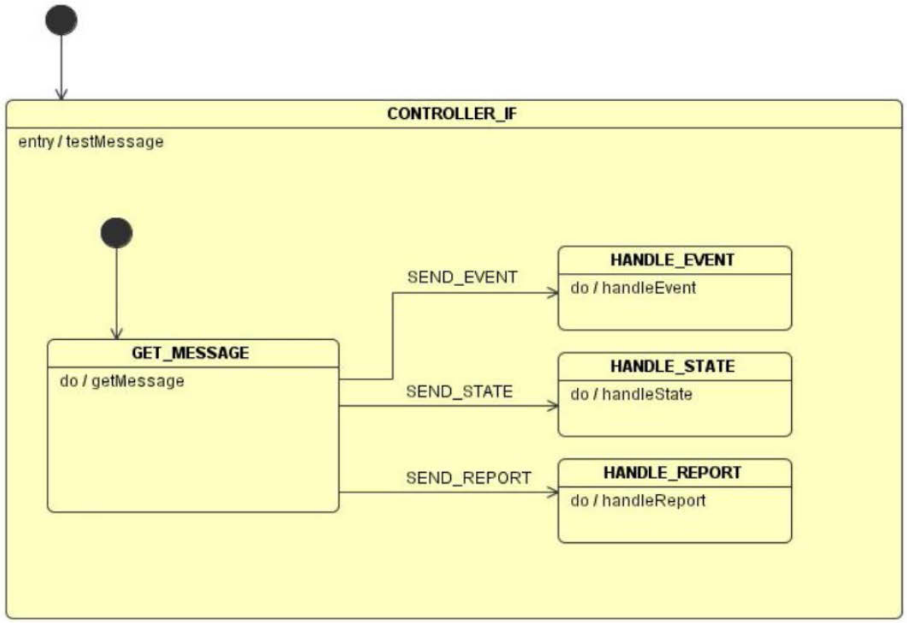
** Source: Marek Tuszynski*

Statechart Framework

- **Based on Miro Samek's Quantum Framework**
- **Standard Event mechanism**
- **Standard Active Object model.**



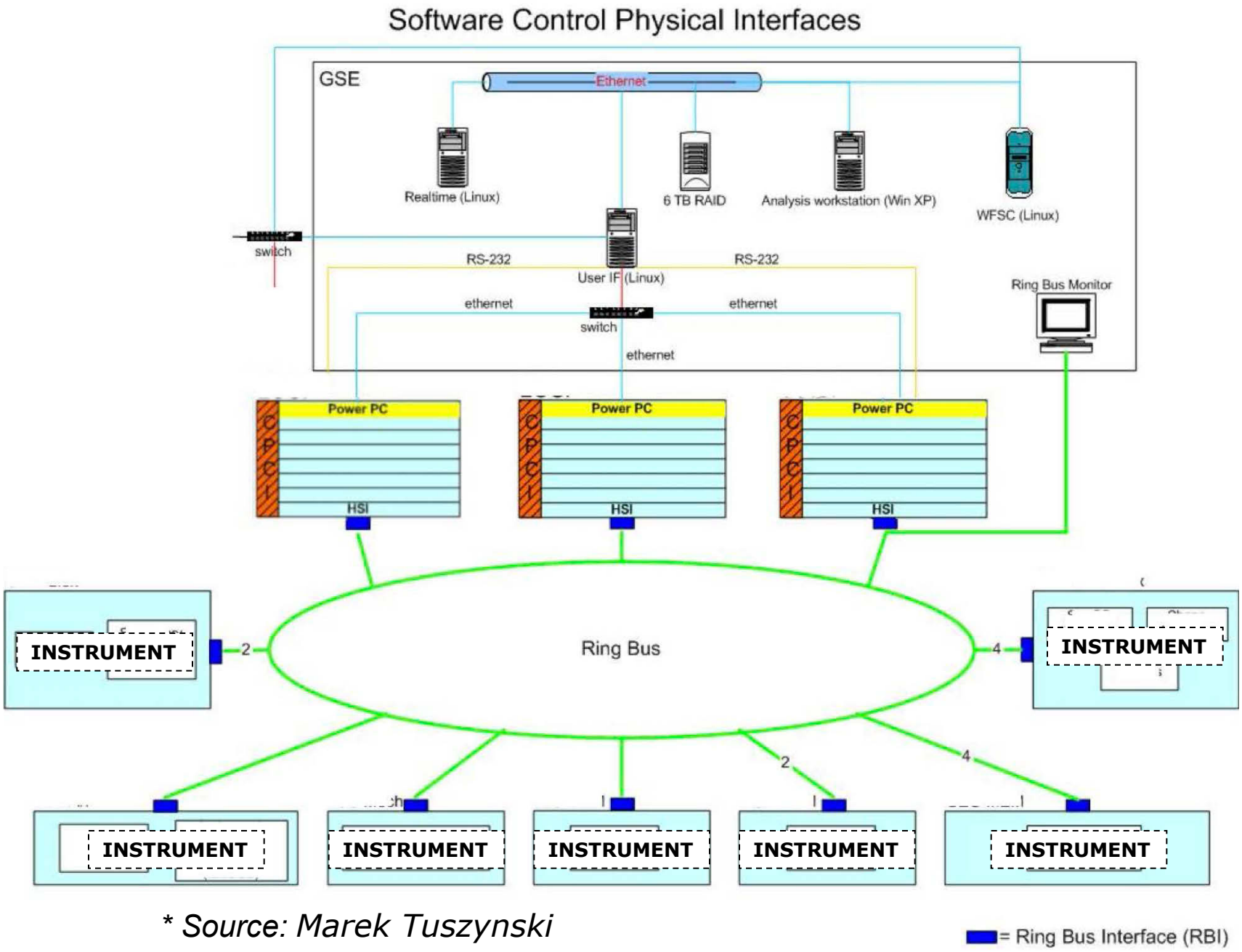
Autocoding the Statecharts



STATECHART

```
#ifndef _imc_h
#define _imc_h

#include <iostream>
#include <cstdio>
#include "port.h"

class ImcImpl;

class Imc : public QActive {

  public:
    Imc(char* objName, ImcImpl* implPtr)
          : QActive((QPseudoState)&Imc::initial)
          , objName(objName)
          , impl(implPtr)
    {}

    void Imc::initial(QEvent const* e);
    QSTATE CONTROLLER_IF_ST(QEvent const *e);
    QSTATE GET_MESSAGE_ST(QEvent const *e);
    QSTATE HANDLE_EVENT_ST(QEvent const *e);

    QSTATE Idle(QEvent const *e);

  private:
    string objName;
    ImcImpl* impl;

};
#endif // _imc_h
```

CODE



** Source: Marek Tuszynski*

RTC FSW to VxWorks

- In migrating an application to a real-time SMP multicore environment, the software developer must understand the lowest common functional units, the interaction between these functional units and the datasets that they are working on.
- Classifying or reclassifying parallel algorithms.
- Identify the needed high-level concurrency design patterns where optimizations can be applied.
- Modifying the RTC configuration mechanism to support deploying components amongst cores … computers …

BACKUP

RTC FSW to VxWorks

- **The RTC FSW code was already a multi-process application, making use of the VxWorks tasks assigned to different rate groups (1Hz, 10Hz … 5kHz).**
- **The implementation of the algorithms remained unchanged.**
- **Implementing task barriers for VxWorks tasks and thread barriers for threads based algorithms and …**
- **Implementing standard time (Unix/POSIX/CCSDS SOIS/IEEE) functions, functions for setting the clock time, synchronizing clocks, propagating time …**

VxWorks Supported Sync Primitives

- **It should be noted that the POSIX Advanced Realtime Threads section defines the following features:**
  - **_POSIX_BARRIERS**
  - **_POSIX_SPIN_LOCKS**
  - **_POSIX_THREAD_CPUTIME**
  - **_POSIX_THREAD_SPORADIC_SERVER**
- **The POSIX *Realtime threads* functions provide the same functionality, but not all operating systems implement them.**
- **VxWorks supports a nominal implementation of POSIX but does not provide the pthread_barrier_wait function.**
- **As such some of this functionality needed to be written and tested and implemented.**

Added Sync Primitives to VxWorks

- **A pure VxWorks API task barrier implementation was written.**
- **A Pthreads thread barrier implementation was written for PORTABILITY – *for applications that may want this … this was not required but I did it anyway*.**
- ***Various UTILITY functions were also written (mostly relating to timing).***